数据库概论——最大并发间隔问题的解决方法比较

刘 茁 1500011438

杨纪翔 1500011342

孙 澎 1500017753

**一、引言**

本报告针对解决最大并发间隔问题的三种常用方法——集合法、游标法和窗口函数法，采用自行生成的多种规模的数据集进行实验，以比较它们的性能。

**二、问题概述**

最大并发间隔问题主要研究以下问题：

在一段时间T内，有若干段并发运行的程序p1, p2, …, pn, 假设程序间互不影响，每个程序都拥有自己的起始、终止时间记录（starttime, endtime），并仅在自己的这两个时间点之间持续运行，在其他时间内不运行。要求在这段时间T内找到一个时间段，使得在这个时间段上正在并发运行的程序数量最大。

**三、方法介绍**

为解决最大并发间隔问题，目前主要有三种常用方法。下文对这三种方法分别进行介绍，随后通过多组变规模的实验比较其性能。为表达直观，本文默认时间点从左向右为从先到后的排列。

1. **集合法**

集合法的思路为，取每个可能时间段中正在并行运行的程序集合，再从中找出程序数量最大的时间段。实际操作时，本文采用的算法选取每个程序的起始时间，对每个起始时间上并行运行的程序数量进行计数，从中找到最大的计数结果所对应的起始时间，即完成了任务。以下解释这样操作的合理性。

由于符合题目要求的最大并发间隔时间段需要包括最多的并行程序，因此它必然符合这样的条件：任何程序的开始或结束行为不会发生在该时间段除了端点之外的内部。因此，假设我们找到了一个符合题目要求的最大并发间隔时间段，若距此时间段左端点最近的起始点刚好位于左端点上，那么这个最大时间段不能向左延伸，因为延伸会导致越过一个起始点，减少一个包括的程序；而如果这个起始点位于左端点左侧，那么将此时间段向左延伸，在越过该起始点前，此时间段包括的程序数量不会改变，因此它仍然为最优。

由此，最大并发间隔时间段一定对应着某个程序的起始点，任何不以程序起始点为起始点的最大并发间隔时间段一定可以延伸到以左侧最近的起始点为起点的更大的时间段，并保持最优性。这样一来，选取可能程序集合的方法就从遍历所有时间段化为遍历所有程序的起始点，而一定存在某个起始点，其与其右侧一个单位的时间点组成的时间段覆盖了最多的并发程序。至此，合理性得证。代码如下：

------------------------------------------------------

WITH TimePoints AS

(

SELECT starttime AS ts FROM dbo.MySessions

),

Counts AS

(

SELECT ts,

(SELECT COUNT(\*)

FROM dbo.MySessions AS S

WHERE P.ts >= S.starttime

AND P.ts < S.endtime) AS concurrent

FROM TimePoints AS P

)

SELECT ts, cnt

FROM Counts

WHERE cnt = (SELECT MAX(concurrent)

FROM Counts)

------------------------------------------------------

由于对每个起始点，算法需要遍历整个程序序列，寻找符合并行条件的程序，故算法的时间复杂度为O(n2)。

1. **游标法**

游标法将所有程序的起始点、终止点排列为一个序列，将每个起始点标记为+1，终止点标记为-1，使用一个游标从左向右，按照时间顺序依次遍历每一个时间点。另使用一个初始值为0的变量a，每次游标遇到一个数b，则对a进行SET a = a + b的更新，并将该值存在对应的时间点处。+1代表一个程序的开始，表示覆盖的并行程序增加了一个；-1表示一个程序结束，代表覆盖的并行程序减少了一个。由此，游标遍历结束后，重新遍历一边所有时间点存的a值，找到最大值所在的时间点，可取该点与其右侧一个时间点间的时间段，即为问题所求的最大并行间隔时间段。代码如下：

---------------------------------------------------

DECLARE

@ts AS datetime,

@type AS int,

@concurrent AS int,

@mx AS int,

@mxts AS int

DECLARE sessions\_cur CURSOR FAST\_FORWARD FOR

SELECT starttime AS ts, +1 AS type

FROM dbo.MySessions

UNION ALL

SELECT endtime, -1

FROM dbo.MySessions

ORDER BY ts, type

OPEN sessions\_cur

FETCH NEXT FROM sessions\_cur

INTO @ts, @type

SET @concurrent = 0,

@mx = 0,

@mxts = 0

WHILE @@FETCH\_STATUS = 0

BEGIN

SET @concurrent = @concurrent + @type

IF @concurrent > @mx SET @mx = @concurrent, @mxts = @ts

FETCH NEXT FROM sessions\_cur

INTO @ts, @type

END

CLOSE sessions\_cur

DEALLOCATE sessions\_cur

SELECT @mxts, @mx

GO

---------------------------------------------------

此算法仅有一层循环，仅需对所有程序的时间点进行顺序遍历，因此时间复杂度为O(n)。

1. **窗口函数法**

窗口函数法原理与游标法十分接近，只是在游标的基础上做了进一步的简化，利用了SQL中的窗口函数机制:

SELECT \*,

SUM(type) OVER(ORDER BY ts, type

ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW) AS cnt

FROM C1

实现与游标方法中“排序后遍历并同步累加”相等价的效果，并将所有的累加结果存放在cnt中。代码如下：

---------------------------------------------------

WITH C1 AS

(

SELECT starttime AS ts, +1 AS type

FROM dbo.MySessions

UNION ALL

SELECT endtime, -1

FROM dbo.MySessions

),

C2 AS

(

SELECT \*,

SUM(type) OVER(ORDER BY ts, type

ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW) AS cnt

FROM C1

)

SELECT ts, cnt

FROM C2

WHERE cnt = (SELECT MAX(cnt)

FROM C2)

---------------------------------------------------

此算法时间复杂度也为O(n)。

**四、实验结果与分析**

本组使用代码随机生成数量为N的数据，分别作为三种方法的输入，记录各方法得出最终结果所需要的时间，并经多次试验取平均值。根据n的不同，将结果列表如下,其中时间均以‘s’为单位：

表1 三种方法的结果统计表

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| N | 10 | 100 | 1000 | 10000 | 30000 | 100000 |
| Set | 0.009 | 0.012 | 0.128 | 4.190 | 65.596 | 402.126 |
| Cursor | 0.010 | 0.010 | 0.039 | 0.246 | 0.688 | 2.209 |
| Window | 0.010 | 0.008 | 0.009 | 0.048 | 0.127 | 0.434 |

从表中：

1. N > 1000部分可以清晰地看出各方法所用时间的翻倍与数据量翻倍的关系，符合我们对各算法复杂度的分析。
2. N < 1000部分，由于数据量过小，因此游标算法与窗口函数方法中的很多副操作占了耗时的较大比例，而这些操作不论对多少数据，操作量都是类似的。因此，N < 1000时，它们的耗时均为0.01s左右。而集合操作由于较为直接，因此在N = 10时甚至比另两种方法耗时更少。

**五、结论**

三种方法解决最大并发间隔问题，拥有不同的时间复杂度。

1. 集合法时间复杂度最高，为O(n2)，仅适合数据量很小时使用，因为此时其效率足够且不需要太多冗余的开销。
2. 游标法和窗口函数法原理相近，时间复杂度均为O(n)，其中窗口函数由于进一步简化了遍历访问数据的操作，因此在处理较大规模的数据方面，比游标法更高效。